

# 20CS2016L Database Systems Lab – B2 URK21CS1128

Ex. No: 06	PL/SQL – FUNCTIONS AND PROCEDURES
Date	

## Objective

To create procedure, function and cursor and perform various operations.

## Description

PL/SQL supports variables, conditions, loops and exceptions, arrays are also supported, though in a somewhat unusual way, involving the use of PL/SQL collections. PL/SQL collections are a slightly advanced topic. Implementations from version 8 of Oracle Database onwards have included features associated with orientation. PL/SQL program units (essentially code containers) can be compiled into the Oracle database. Programmers can thus embed PL/SQL units of functionality into the database directly. They also can write scripts containing PL/SQL program units that can be read into the database using the Oracle SQL \*Plus tool.

Once the program units have been stored into the database, they become available for execution at a later time. While programmers can readily embed Data Manipulation Language (DML) statements directly into their PL/SQL code using straight forward SQL statements, Data Definition Language (DDL) requires more complex "Dynamic SQL" statements to be written in the PL/SQL code. However, DML statements underpin the majority of PL/SQL code in typical software applications.

In the case of PL/SQL dynamic SQL, early versions of the Oracle Database required the use of a complicated Oracle DBMS\_SQL package library. More recent versions have however introduced a simpler "Native Dynamic SQL", along with an associated EXECUTE IMMEDIATE syntax. Oracle Corporation customarily extends package functionality with each successive release of the Oracle Database.

## Detailed Procedure:

Anonymous blocks form the basis of the simplest PL/SQL code, and have the following structure:

<<label>>

DECLARE

TYPE / item / FUNCTION / PROCEDURE declarations

# 20CS2016L Database Systems Lab – B2 URK21CS1128

---

BEGIN

Statements

EXCEPTION

EXCEPTION handlers

END label;

The <<label>> and the DECLARE and EXCEPTION sections are optional.

Exceptions, errors which arise during the execution of the code, have one of two types:

1. Predefined exceptions
2. User-defined exceptions.

User-defined exceptions are always raised explicitly by the programmers, using the RAISE or RAISE\_APPLICATION\_ERROR commands, in any situation where they have determined that it is impossible for normal execution to continue. RAISE command has the syntax:

RAISE <exception name>. Oracle Corporation has pre-defined several exceptions like NO\_DATA\_FOUND, TOO\_MANY\_ROWS, *etc.* Each exception has a SQL Error Number and SQL Error Message associated with it. Programmers can access these by using the SQLCODE and SQLERRM functions. The DECLARE section defines and (optionally) initializes variables. If not initialized specifically, they default to NULL.

For example:

DECLARE

number1 NUMBER(2);

number2 NUMBER(2) := 17;            -- value default

text1 VARCHAR2 (12):= 'Hello world';

text2 DATE        := SYSDATE;        -- current date and time

BEGIN

SELECT street\_number

INTO number1

## 20CS2016L Database Systems Lab – B2 URK21CS1128

---

FROM address

WHERE name = 'INU';

END;

The symbol := functions as an assignment operator to store a value in a variable.

The major datatypes in PL/SQL include NUMBER, INTEGER, CHAR, VARCHAR2, DATE, TIMESTAMP, TEXT *etc.*

### Functions

Functions in PL/SQL are a collection of SQL and PL/SQL statements that perform a task and should return a value to the calling environment. User defined functions are used to supplement the many hundreds of functions built in by Oracle.

There are two different types of functions in PL/SQL. The traditional function is written in the form:

```
CREATE OR REPLACE FUNCTION <function_name> [(input/output variable declarations)]  
RETURN return_type
```

```
[AUTHID <CURRENT USER | DEFINER>] <IS|AS>
```

```
    [declaration block]
```

```
BEGIN
```

```
    <PL/SQL block WITH RETURN statement>
```

```
    RETURN <return_value>;
```

```
[EXCEPTION
```

```
    EXCEPTION block]
```

```
    RETURN <return_value>;
```

```
END;
```

Pipelined Table Functions are written in the form:

## 20CS2016L Database Systems Lab – B2 URK21CS1128

---

```
CREATE OR REPLACE FUNCTION <function_name> [(input/output variable declarations)]
RETURN return_type

[AUTHID <CURRENT USER | DEFINER>] [<AGGREGATE | PIPELINED>] <IS|USING>

    [declaration block]

BEGIN

    <PL/SQL block WITH RETURN statement>

    PIPE ROW <return type>;

    RETURN;

[EXCEPTION

    EXCEPTION block]

    PIPE ROW <return type>;

    RETURN;

END;
```

There are three types of parameters: IN, OUT and IN OUT. An IN parameter is used as input only. An IN parameter is passed by copy and thus cannot be changed by the called program. An OUT parameter is initially NULL. The program assigns the parameter a value and that value is returned to the calling program. An IN OUT parameter may or may not have an initial value. That initial value may or may not be modified by the called program. Any changes made to the parameter are returned to the calling program by default by copying but, with the NOCOPY hint may be passed by reference.

### Procedures

Procedures are similar to Functions; in that they can be executed to perform work. The primary difference is that procedures cannot be used in a SQL statement and although they can have multiple out parameters they do not "RETURN" a value.

Procedures are traditionally the workhorse of the coding world and functions are traditionally the smaller, more specific pieces of code. PL/SQL maintains many of the distinctions between functions and procedures found in many general-purpose programming languages, but in addition, functions can be called from SQL, while procedures cannot.

# 20CS2016L Database Systems Lab – B2 URK21CS1128

---

**These are the real time usages of PL/SQL.**

- **Block Structures:** PL SQL consists of blocks of code, which can be nested within each other. Each block forms a unit of a task or a logical module. PL/SQL Blocks can be stored in the database and reused.
- **Procedural Language Capability:** PL SQL consists of procedural language constructs such as conditional statements (if else statements) and loops like (FOR loops).
- **Better Performance:** PL SQL engine processes multiple SQL statements simultaneously as a single block, thereby reducing network traffic.
- **Error Handling:** PL/SQL handles errors or exceptions effectively during the execution of a PL/SQL program. Once an exception is caught, specific actions can be taken depending upon the type of the exception or it can be displayed to the user with a message.

## **Questions:**

1. Write a PL/SQL function named `getUserEmail` that takes a user's name as input and returns their email address as output. Assume you have a table named `users` with the following columns: `UserID`, `Name`, `Email`, `Password`, and `Phone`. The function should use the input name to retrieve the corresponding email address from the table and return it.

```
Query: -- Declare a variable to store the result
DECLARE
    userEmail VARCHAR2(255);
BEGIN
    userEmail := getUserEmail('John Smith');
    IF userEmail IS NOT NULL THEN
        DBMS_OUTPUT.PUT_LINE('User Email: ' || userEmail);
    ELSE
        DBMS_OUTPUT.PUT_LINE('User not found.');
```

END IF;

END;

/

## 20CS2016L Database Systems Lab – B2 URK21CS1128

---

### Output:

```
SQL> CREATE OR REPLACE FUNCTION getUserEmail(userName IN VARCHAR2) RETURN VARCHAR2 IS
 2   userEmail VARCHAR2(255);
 3 BEGIN
 4   SELECT Email INTO userEmail
 5   FROM User_1128
 6   WHERE Name = userName;
 7   RETURN userEmail;
 8 EXCEPTION
 9   WHEN NO_DATA_FOUND THEN
10     RETURN NULL; -- User not found
11 END;
12 /

Function created.
```

2. Write a PL/SQL procedure named changePhoneNumber that takes a user's name and a new phone number as input and updates the phone number in the users table. Assume you have a table named users with the following columns: UserID, Name, Email, Password, and Phone. The procedure should use the provided name to locate the user and update their phone number with the new one.

Query: CREATE OR REPLACE PROCEDURE changePhoneNumber(userName IN VARCHAR2, newPhoneNumber IN VARCHAR2) IS

BEGIN

UPDATE User\_1128

SET Phone = newPhoneNumber

WHERE Name = userName;

COMMIT;

END;

/

## 20CS2016L Database Systems Lab – B2 URK21CS1128

---

Output:

```
SQL> CREATE OR REPLACE PROCEDURE changePhoneNumber(userName IN VARCHAR2, newPhoneNumber IN VARCHAR2) IS
2 BEGIN
3   UPDATE User_1128
4     SET Phone = newPhoneNumber
5     WHERE Name = userName;
6   COMMIT;
7 END;
8 /

Procedure created.
```

3. Write a PL/SQL function named `getEventDescription` that takes an event name as input and returns its description as output. Assume you have a table named `events` with the following columns: `EventID`, `Name`, `Date`, `Time`, `VenueID`, and `Description`. The function should use the input event name to retrieve the corresponding event description from the table and return it.

Query: `CREATE OR REPLACE FUNCTION getEventDescription(eventName IN VARCHAR2) RETURN VARCHAR2 IS`

`eventDescription VARCHAR2(1000); -- Adjust the size as needed`

`BEGIN`

`SELECT Description INTO eventDescription`

`FROM Event_1128`

`WHERE Name = eventName;`

`RETURN eventDescription;`

`EXCEPTION`

`WHEN NO_DATA_FOUND THEN`

`RETURN NULL; -- Event not found`

`END;`

`/`

## 20CS2016L Database Systems Lab – B2 URK21CS1128

---

Output:

```
SQL> CREATE OR REPLACE FUNCTION getEventDescription(eventName IN VARCHAR2) RETURN VARCHAR2 IS
  2   eventDescription VARCHAR2(1000); -- Adjust the size as needed
  3 BEGIN
  4   SELECT Description INTO eventDescription
  5   FROM Event_1128
  6   WHERE Name = eventName;
  7   RETURN eventDescription;
  8 EXCEPTION
  9   WHEN NO_DATA_FOUND THEN
 10    RETURN NULL; -- Event not found
 11 END;
 12 /
```

Function created.

4. Write a PL/SQL procedure named `updateEventVenue` that takes an event name and a new venue ID as input and updates the venue ID in the events table. Assume you have a table named `events` with the following columns: `EventID`, `Name`, `Date`, `Time`, `VenueID`, and `Description`. The procedure should use the provided event name to locate the event and update its venue ID with the new one.

Query: `CREATE OR REPLACE PROCEDURE updateEventVenue(eventName IN VARCHAR2, newVenueID IN NUMBER) IS`

`BEGIN`

`UPDATE Event_1128`

`SET VenueID = newVenueID`

`WHERE Name = eventName;`

`COMMIT;`

`END;`

`/`



## 20CS2016L Database Systems Lab – B2 URK21CS1128

---

Output:

```
SQL> CREATE OR REPLACE PROCEDURE updateEventVenue(eventName IN VARCHAR2, newVenueID IN
NUMBER) IS
2 BEGIN
3     UPDATE Event_1128
4     SET VenueID = newVenueID
5     WHERE Name = eventName;
6     COMMIT;
7 END;
8 /

Procedure created.
```

5. Write a PL/SQL function named `getVenueAddress` that takes a venue name as input and returns its address as output. Assume you have a table named `venues` with the following columns: `VenueID`, `Name`, `Address`, `City`, `State`, and `Country`. The function should use the input venue name to retrieve the corresponding venue address from the table and return it.

Query: `CREATE OR REPLACE FUNCTION getVenueAddress(venueName IN VARCHAR2) RETURN VARCHAR2 IS`

`venueAddress VARCHAR2(255);`

`BEGIN`

`SELECT Address INTO venueAddress`

`FROM Venue_1128`

`WHERE Name = venueName;`

`RETURN venueAddress;`

`EXCEPTION`

`WHEN NO_DATA_FOUND THEN`

`RETURN NULL; -- Venue not found`

`END;`

`/`

## 20CS2016L Database Systems Lab – B2 URK21CS1128

---

Output:

```
SQL> CREATE OR REPLACE PROCEDURE updateEventVenue(eventName IN VARCHAR2, newVenueID IN
NUMBER) IS
2 BEGIN
3     UPDATE Event_1128
4     SET VenueID = newVenueID
5     WHERE Name = eventName;
6     COMMIT;
7 END;
8 /

Procedure created.
```

6. Write a PL/SQL procedure named updateVenueCity that takes a venue name and a new city as input and updates the city in the venues table. Assume you have a table named venues with the following columns: VenueID, Name, Address, City, State, and Country. The procedure should use the provided venue name to locate the venue and update its city with the new one.

Query: CREATE OR REPLACE PROCEDURE updateVenueCity(venueName IN VARCHAR2, newCity IN VARCHAR2) IS

BEGIN

UPDATE Venue\_1128

SET City = newCity

WHERE Name = venueName;

COMMIT;

END;

/

Output:

```
SQL> CREATE OR REPLACE PROCEDURE updateVenueCity(venueName IN VARCHAR2, newCity IN VARC
HAR2) IS
2 BEGIN
3     UPDATE Venue_1128
4     SET City = newCity
5     WHERE Name = venueName;
6     COMMIT;
7 END;
8 /

Procedure created.
```

## 20CS2016L Database Systems Lab – B2 URK21CS1128

---

7. Write a PL/SQL function named `getVenueState` that takes a venue name as input and returns its state as output. Assume you have a table named `venues` with the following columns: `VenueID`, `Name`, `Address`, `City`, `State`, and `Country`. The function should use the input venue name to retrieve the corresponding venue state from the table and return it.

Query: `CREATE OR REPLACE FUNCTION getVenueState(venueName IN VARCHAR2) RETURN VARCHAR2 IS`

`venueState VARCHAR2(50);`

`BEGIN`

`SELECT State INTO venueState`

`FROM Venue_1128`

`WHERE Name = venueName;`

`RETURN venueState;`

`EXCEPTION`

`WHEN NO_DATA_FOUND THEN`

`RETURN NULL; -- Venue not found`

`END;`

`/`

Output:

```
SQL> CREATE OR REPLACE FUNCTION getVenueState(venueName IN VARCHAR2) RETURN VARCHAR2 IS
  2     venueState VARCHAR2(50);
  3 BEGIN
  4     SELECT State INTO venueState
  5     FROM Venue_1128
  6     WHERE Name = venueName;
  7     RETURN venueState;
  8 EXCEPTION
  9     WHEN NO_DATA_FOUND THEN
 10         RETURN NULL; -- Venue not found
 11 END;
 12 /

Function created.
```

## 20CS2016L Database Systems Lab – B2 URK21CS1128

---

8. Write a PL/SQL function named `getTicketPrice` that takes a ticket ID as input and returns the price of the ticket as output. Assume you have a table named `tickets` with the following columns: `TicketID`, `EventID`, `UserID`, `SeatNumber`, `Price`, and `Status`. The function should use the input ticket ID to retrieve the corresponding ticket price from the table and return it.

```
Query: CREATE OR REPLACE FUNCTION getTicketPrice(ticketID IN NUMBER)
RETURN NUMBER IS

    ticketPrice NUMBER;

BEGIN

    SELECT Price INTO ticketPrice

    FROM Ticket_1128

    WHERE TicketID = ticketID;

    RETURN ticketPrice;

EXCEPTION

    WHEN NO_DATA_FOUND THEN

        RETURN NULL; -- Ticket not found

END;

/
```

Output:

```
SQL> CREATE OR REPLACE FUNCTION getTicketPrice(ticketID IN NUMBER) RETURN NUMBER IS
 2     ticketPrice NUMBER;
 3 BEGIN
 4     SELECT Price INTO ticketPrice
 5     FROM Ticket_1128
 6     WHERE TicketID = ticketID;
 7     RETURN ticketPrice;
 8 EXCEPTION
 9     WHEN NO_DATA_FOUND THEN
10         RETURN NULL; -- Ticket not found
11 END;
12 /

Function created.
```

## 20CS2016L Database Systems Lab – B2 URK21CS1128

---

9. Write a PL/SQL procedure named `updateTicketStatus` that takes a ticket ID and a new status as input and updates the status in the tickets table. Assume you have a table named `tickets` with the following columns: `TicketID`, `EventID`, `UserID`, `SeatNumber`, `Price`, and `Status`. The procedure should use the provided ticket ID to locate the ticket and update its status with the new one.

Query: `CREATE OR REPLACE PROCEDURE updateTicketStatus(ticketID IN NUMBER, newStatus IN VARCHAR2) IS`

`BEGIN`

`UPDATE Ticket_1128`

`SET Status = newStatus`

`WHERE TicketID = ticketID;`

`COMMIT;`

`END;`

`/`

Output:

```
SQL> CREATE OR REPLACE PROCEDURE updateTicketStatus(ticketID IN NUMBER, newStatus IN VA
RCHAR2) IS
2 BEGIN
3 UPDATE Ticket_1128
4 SET Status = newStatus
5 WHERE TicketID = ticketID;
6 COMMIT;
7 END;
8 /

Procedure created.
```

10. Write a PL/SQL function named `getTotalTicketsForEvent` that takes an event ID as input and returns the total number of tickets booked for that event as output. Assume you have a table named `tickets` with the following columns: `TicketID`, `EventID`, `UserID`, `SeatNumber`, `Price`, and `Status`. The function should use the input event ID to count the number of tickets with the specified event ID in the table and return that count.

## 20CS2016L Database Systems Lab – B2 URK21CS1128

---

Query: CREATE OR REPLACE FUNCTION getTotalTicketsForEvent(eventID IN NUMBER) RETURN NUMBER IS

totalTickets NUMBER;

BEGIN

SELECT COUNT(\*) INTO totalTickets

FROM Ticket\_1128

WHERE EventID = eventID;

RETURN totalTickets;

END;

/

Output:

```
SQL> CREATE OR REPLACE FUNCTION getTotalTicketsForEvent(eventID IN NUMBER) RETURN NUMBER IS
2   totalTickets NUMBER;
3 BEGIN
4   SELECT COUNT(*) INTO totalTickets
5   FROM Ticket_1128
6   WHERE EventID = eventID;
7   RETURN totalTickets;
8 END;
9 /

Function created.
```

**Result:**

The given queries were executed successfully using pl/sql – functions and procedures.